EMBEDDED
LINUX
CONFERENCE

Golang Support in Yocto Project

Vyacheslav Yurkov
Lukas Funke

THE LINUX FOUNDATION

#EmbeddedOSSummit

About us

- Use Linux since 2000.
- 20+ years of experience in different industries
- Regular contributor to open source projects (notably Linux Kernel, TF-A, Yocto Project, and others)
- Maintainer of OverlayFS classes in Openembedded Core

Vyacheslav Yurkov
(Slava)

Contacts:

Email: uvv.mail@gmail.com

LinkedIn: https://www.linkedin.com/in/vyacheslav-yurkov-2504ba18/

Github: https://github.com/UVV-gh

Gitlab: https://gitlab.com/UVV-gh

About us

- 10+ years of experience with Linux and embedded systems
- Contributor to open source projects (Yocto Project, U-Boot and others)
- Currently working @Weidmueller

Lukas Funke

Contacts:

Email: lukas.funke@gmail.com, lukas.funke@weidmueller.com

Github: https://github.com/luckyfunky

Weidmüller

Agenda

- How C and C++ software is built with Yocto
- Dependencies in Go
- Original support in Yocto
- Current state
- Future plans

Yocto terminology, bitbake

- Bitbake - Build System
- Bitbake - Task scheduler and execution engine
- Bitbake parses configuration data and instructions (recipes, see next slides)

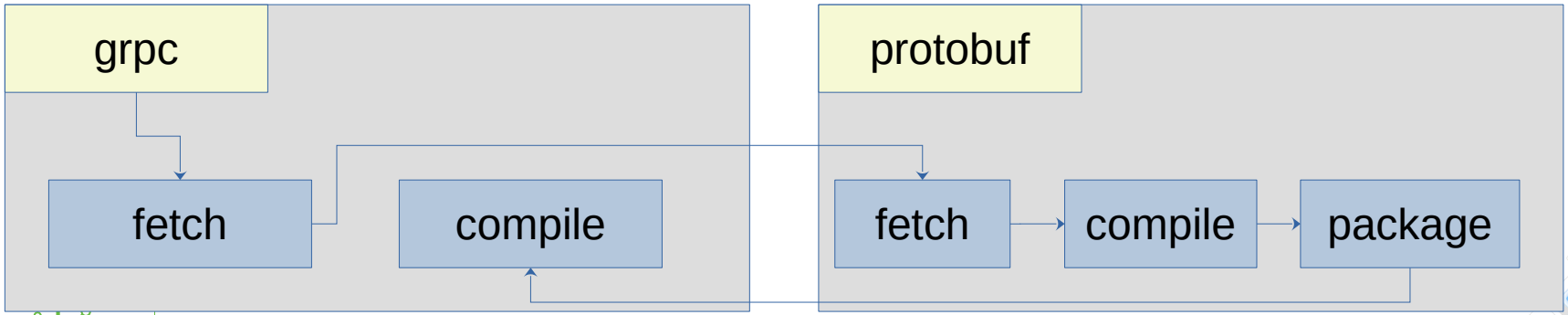
- It is written in Python

Yocto terminology, recipes

- A text file, which is parsed by *Bitbake,* containing instructions how to build each software component, with the extension *.bb*
- A recipe describes where you get source code and which patches to apply
- Recipes describe dependencies for libraries or for other recipes
- They have a specific syntax (reminds a mix between bash and python)
- *Bitbake* can build individual recipes resolving all required dependencies or even a specific *task* from a recipe (see next slide)

Yocto terminology, tasks

- The build process implemented by a recipe is split in several tasks
- Tasks are units of execution for *BitBake*
- Each task performs a specific step in the build
- Examples: fetch, patch, configure, compile, package
- Tasks have dependencies on other tasks (even tasks of other recipes)
- Task states are cached in order to be re-used (shared state cache)



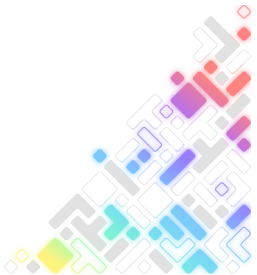
C or C++ build systems

- Makefile
- Autotools
- CMake
- Meson

- Bazel
- Gradle
- SCons
- QMake

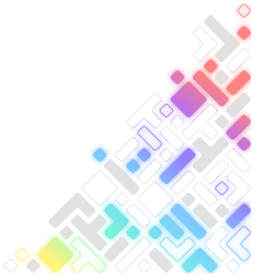
C or C++ build configuration, cmake (1)

```
cmake_minimum_required(VERSION 3.25)
project(MyCppExampleProject LANGUAGES CXX)

find_package(Boost 1.78 REQUIRED COMPONENTS filesystem)

add_executable(MyExample main.cpp)

target_link_libraries(MyExample PRIVATE Boost::filesystem)

install(TARGETS MyExample RUNTIME DESTINATION bin)
```

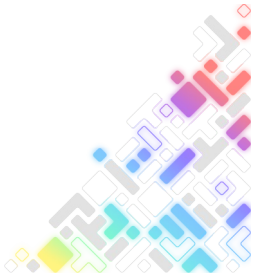
C or C++ build configuration, cmake (2)

```
% mkdir build
% cmake ..
% cmake --build .
% cmake --install .
```



C or C++ build configuration, Yocto recipe (1)

```
DESCRIPTION = "Simple helloworld application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "\
  file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302
"
SRC_URI = "file://helloworld.cpp"
S = "${WORKDIR}"
DEPENDS += "boost"
RDEPENDS:${PN} += "boost-filesystem"

do_compile() {
  ${CC} ${LDFLAGS} helloworld.cpp -o helloworld
}

do_install() {
  install -d ${D}${bindir}
  install -m 0755 helloworld ${D}${bindir}
}
```

C or C++ build configuration, Yocto recipe (2)

```
DESCRIPTION = "Simple helloworld application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "\
  file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302
"
SRC_URI = "file://helloworld.cpp"
S = "${WORKDIR}"
DEPENDS += "boost"
RDEPENDS:${PN} += "boost-filesystem"

do_compile() {
  ${CC} ${LDFLAGS} helloworld.cpp -o helloworld
}

do_install() {
  install -d ${D}${bindir}
  install -m 0755 helloworld ${D}${bindir}
}
```

Build-time dependencies

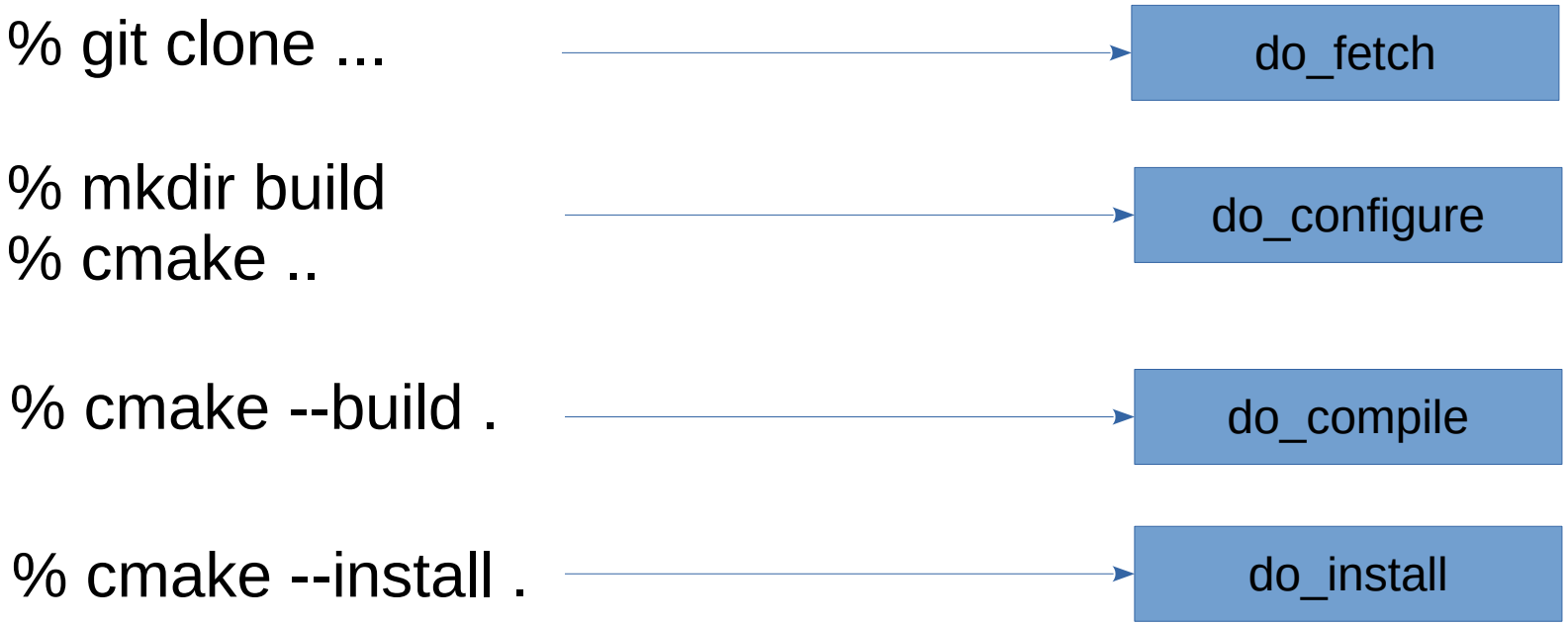
Run-time dependencies

Yocto tasks

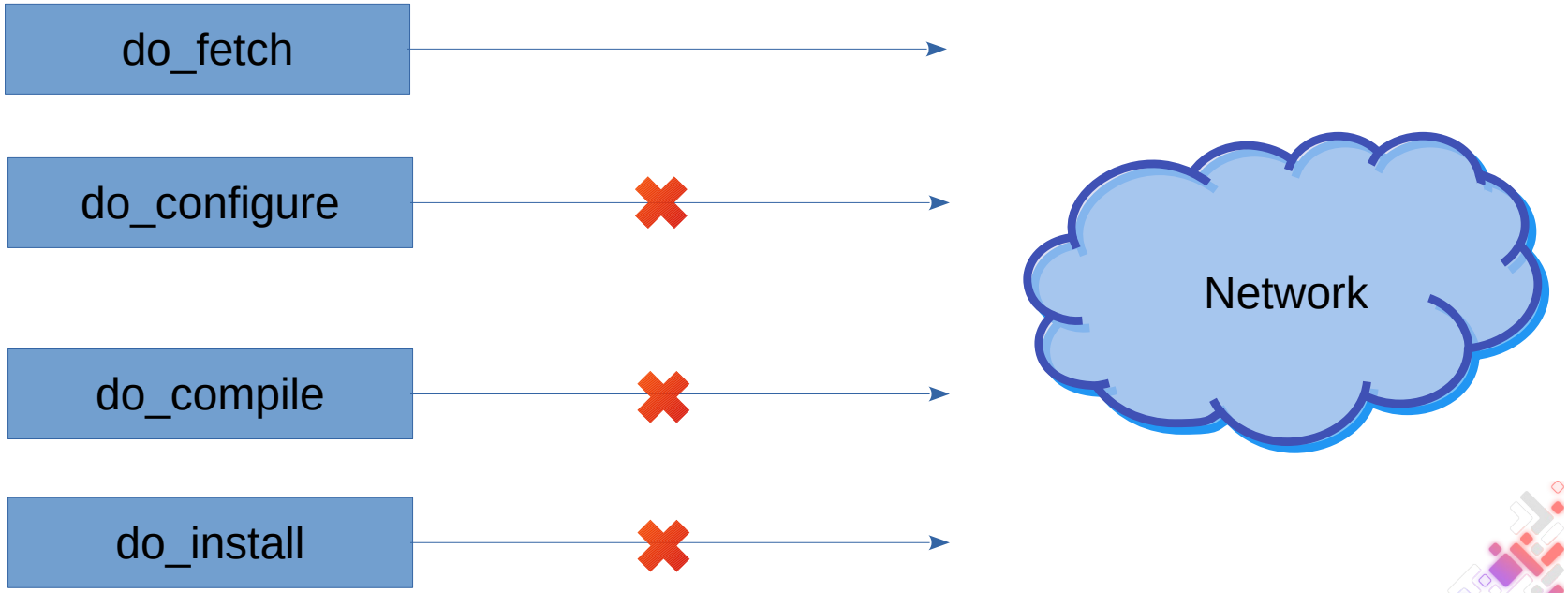
C or C++ build configuration, Yocto recipe (3)

```
DESCRIPTION = "Simple helloworld application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "\
  file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302
"
SRC_URI = "file://helloworld.cpp file://CMakeLists.txt"
S = "${WORKDIR}"
DEPENDS += "boost"
RDEPENDS:${PN} += "boost-filesystem"

inherit cmake
```

C or C++ build configuration, Yocto

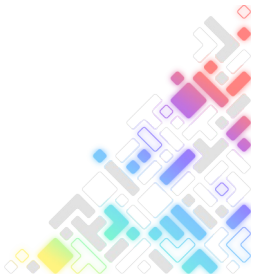


Yocto tasks and network (1)



Yocto tasks and network (2)

- All sources are downloaded with do_fetch() task, URLs are in SRC_URI
- Downloaded artifacts are preserved to speed up subsequent builds (source mirror)
- Network access outside do_fetch() task is possible only when explicitly enabled
- All this allows to achieve reproducible builds and reduce build system complexity



Original support in oe-core (1)

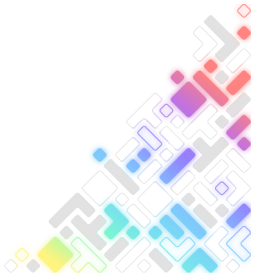
commit 78615e9260fb5d6569de4883521b049717fa4340
Author: Khem Raj <raj.khem@gmail.com>
Date: Tue Mar 7 22:40:22 2017 -0800

go: Add recipes for golang compilers and tools

* This is converging the recipes for go from
  meta-virtualization and oe-meta-go

* Add recipes for go 1.7

* go.bbclass is added to ease out writing
  recipes for go packages



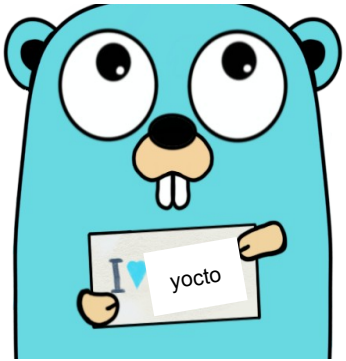
Original support in oe-core (2)

commit 11c2b06ac98cc5064640705712bffa156519f450
Author: Otavio Salvador <otavio.salvador@gmail.com>
Date: Thu May 21 23:22:34 2020 -0300

    go-mod.bbclass: Add class for `go mod` support

...

‘Go’ support in Yocto



- A short overview of *go* modularization concept
  - *Go* applications are **modules**
  - A module is a **collection of packages** which is versioned and released together
  - A module is declared in a **‘go.mod’** package file
  - Modules can **import** other modules/packages

like
`#include <quote.h>`

hello.go

```
package hello

import "rsc.io/quote"

func Hello() string {
    return quote.Hello()
}
```

go.mod

```
module example.com/hello

go 1.12

require rsc.io/quote v1.5.2
```

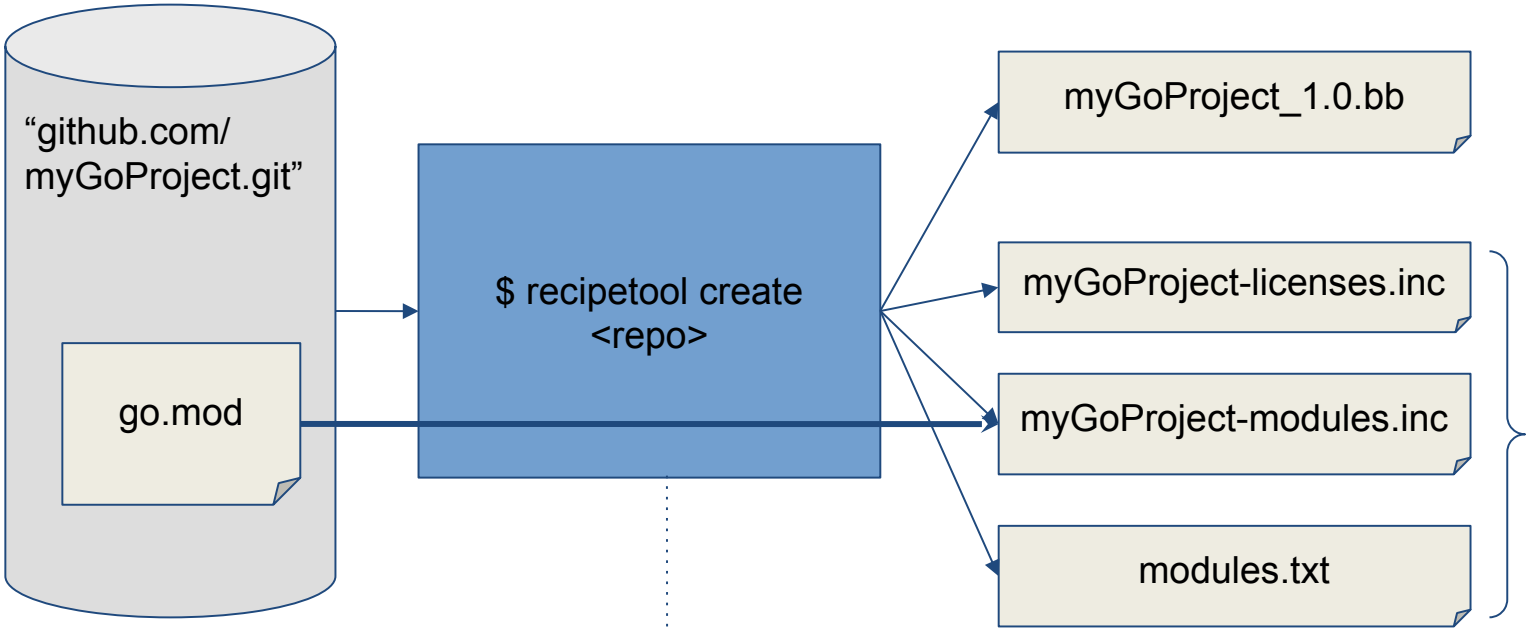
How it worked

- Originally network access was allowed in previous versions
- You enable network access for do_compile() task:
  do_compile[network] = “1”
- You keep all vendor dependencies in your project repository

‘Go’ support in Yocto

- The challenge:
  - Because dependency management of go is **built into the language** and its runtime, **Yocto is unaware of golang’s dependencies**.
  - Thus, **Yocto itself cannot**
    - **fetch, patch or archive** the dependencies
    - deploy the dependencies licenses
    - check CVE of dependencies
  - (go) modules may not be around forever
- How to deal with this problem?

Proposed workflow with ‘recipetool’



‘Go’ support in Yocto - Addressing the challenges

- Fetch
  - Translate the module import-path to a repository URL since there may be **no 1:1 mapping**
    - Example: `go.mozilla.org/pkcs7 => github.com/mozilla-services/pkcs7`
  - Translate the **version** into a **commit hash**
    - Example: `github.com/ghodss/yaml@v1.0.0 => 0ca9ea5df5451ffdf184b4428c902747c2c11cd7`
  - Several sources have to be queried for correct translation
    - golang proxy
    - go-import http statement
    - regex-translation

‘Go’ support in Yocto - Addressing the challenges

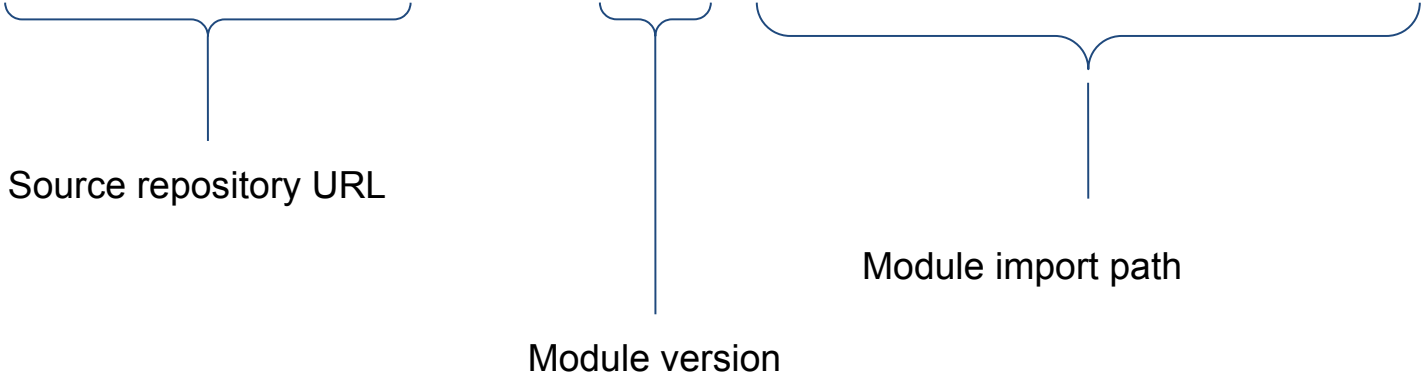
- Vendoring
  - Unpack the fetched dependencies such that we trick *go* into thinking all dependencies were **vendored**
    - Parse *module.txt* in order isolate the correct packages and remove anything else
    - Copy license files from modules root path
    - Handle **replace**, **embed** and **build instruction** statements

Generated <recipe>.bb

```
...
LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://src/${GO_IMPORT}/LICENSE;md5=8f8bc924cf73f6a32381e5fd4c58d603"

SRC_URI = "git://${GO_IMPORT};destsuffix=git/src/${GO_IMPORT};nobranch=1;name=${BPN};protocol=https \
           file://modules.txt \
           "

# Modify these as desired
PV = "1.0+git"
SRCREV = "1fb6775d8804dda106fccace673c66bb187c3e57"

S = "${WORKDIR}/git"

LICENSE += " & ${GO_MOD_LICENSES}"
require ${BPN}-licenses.inc
require ${BPN}-modules.inc
GO_IMPORT = "<Module name>"
SRCREV_FORMAT = "${BPN}"

inherit go-vendor
```

Anatomy of ${PN}-modules.inc

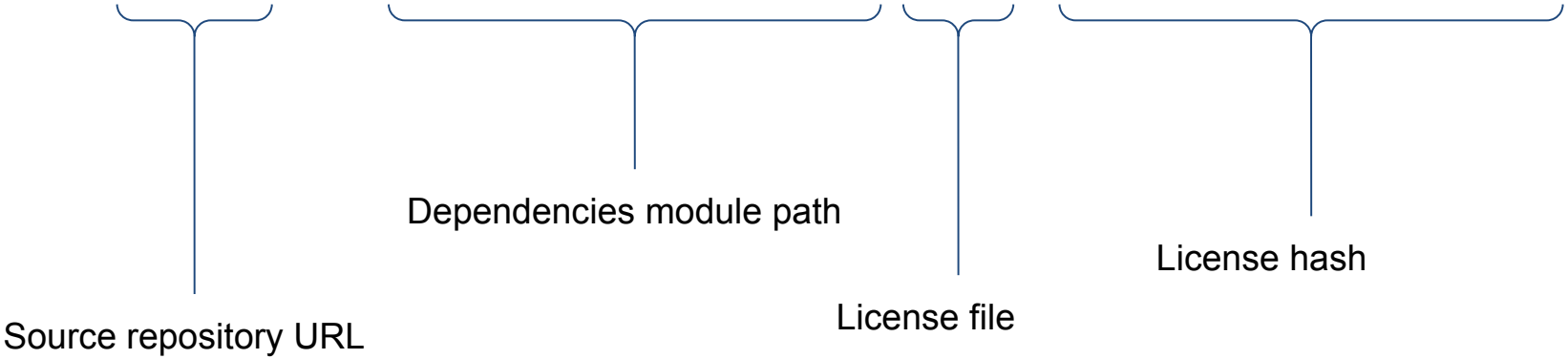
```
SRC_URI += "${GO_DEPENDENCIES_SRC_URI}"      - Add dependencies to the metadata SRC_URI

GO_DEPENDENCIES_SRC_URI = "\
...
${@go_src_uri('github.com/google/uuid','v1.3.0')} \
...
${@go_src_uri('github.com/go-redis/redis','v7.3.0',path='github.com/go-redis/redis/v7'...)}
```




```
# github.com/go-redis/redis/v7@v7.3.0 => 8f52186ce7bae8731c03bf6b2f1c54f375d6cd19
```

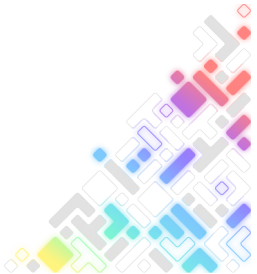
Anatomy of ${PN}-licenses.inc

```
GO_MOD_LICENSES = "Apache-2.0 & BSD-2-Clause & BSD-3-Clause & ISC & MIT" - Licenses introduced by dependencies

...
LIC_FILES_CHKSUM  += "${VENDORED_LIC_FILES_CHKSUM}"
VENDORED_LIC_FILES_CHKSUM = "\
    file://src/${GO_IMPORT}/vendor/github.com/AlecAivazis/survey/v2/LICENSE;md5=615828847c7140c5e6bac90e11888eb5
...
```



Conclusion

- Yocto can make use go **vendoring** in order to handle module dependencies
- We are **independent of additional infrastructure** such as *proxy.golang.org*
- Using the vendoring approach we enable reproducible builds and can minimize supply chain attacks
- Using our approach we can include dependency licenses into the final software product in order fulfill license obligations



Future plans

- CVE check for dependencies
- Automatic dependency update (like rust does it)
- Support private modules (ssh protocol)
- Fix *devtool* workflow

References

- https://docs.yoctoproject.org/ref-manual/tasks.html
- https://bootlin.com/training/yocto/
- https://go.dev/ref/mod
- https://pretalx.com/yocto-project-summit-2022-05/talk/WGF7YE/
- https://lists.openembedded.org/g/openembedded-core/topic/89464905
- https://lists.openembedded.org/g/openembedded-architecture/topic/90782883

Thanks and acknowledgments

- Bruce Ashfield
- Weidmüller Team-OS

EMBEDDED LINUX CONFERENCE